

APPLICATION FOR UNITED STATES LETTERS OF PATENT

FOR

**METHOD AND SYSTEM FOR ACCESS AND DISPLAY OF DATA FROM  
LARGE DATA SETS**

Inventor: **D. Thomas Predovic**

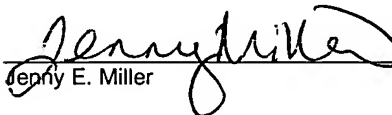
Prepared by:

Blakely Sokoloff Taylor & Zafman, LLP  
12400 Wilshire Boulevard, 7th Floor  
Los Angeles, California 90025  
(206) 292-8600

"Express Mail" Label Number EL861982848US

Date of Deposit December 18, 2001

*I hereby certify that this paper or fee is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to the Assistant Commissioner for Patents, Box Patent Application, Washington, D.C. 20231.*

 12/18/01  
Jenny E. Miller Date

# **METHOD AND SYSTEM FOR ACCESS AND DISPLAY OF DATA FROM LARGE DATA SETS**

## **BACKGROUND OF THE INVENTION**

5

### **1. Field of the Invention**

The present invention concerns databases in general, and, in particular, a method for access and display of data from large data sets in which the large data sets are portioned into a plurality of buckets, each of which contain a small subset of records in the large data set.

10

### **2. Background Information**

During the past decade, the use of databases that store very large amounts of data has become increasingly prevalent. This is due, in part, to the availability of both computer hardware and database software resources to support these large databases. Prior to the present widespread availability of these resources, data was typically stored using "flat-file" databases or data systems running on mainframes or standalone computers using storage schemes that either supported limited sizes, or provided limited real-time access to the data (e.g., data systems that stored data on tapes). Oftentimes, an enterprise organization had various types of data stored on separate machines that did not provide for easy remote access. In contrast, today's IT environments often involve the use of huge centralized repositories in which data for an entire enterprise are stored, wherein the repositories may be accessed from remote clients connected to them via LANs, WANs, or even over the Internet.

15

20

25

The majority of the large databases in operation today are RDBMS (relational database management system) databases. Furthermore, most of these RDBMS

databases are SQL- (structured query language) based databases. These SQL databases run on database software provided by various vendors, including Oracle (Oracle 8i and 9i), Microsoft (SQL Server 7), IBM (DB2), Sybase, and Informix. RDBMS databases are usually run on one or more networked database servers and are accessed by client machines connected to the database server(s) via a computer network. On some n-tier architectures, there are one or more middle tiers (e.g., application servers) that sit between a "backend" database server and the clients. A typical 2-tier architecture is illustrated in FIGURE 1, wherein a database 10 hosted by a database server 12 may be accessed by a client machine 14 via a network 16. Modern SQL RDBMS databases enable a multitude of client users to concurrently access (i.e., Insert, Update and Delete) data using appropriate client-side software (or through middleware running on an application server), such as client applications that provide a graphical user interface (GUI) that allows users to interactively access database data.

In RDBMS databases, data are stored in tables in accordance with a database "schema," which defines storage parameters (metadata) that define structures for each of the tables stored in the database, various data constraints, relationships between tables, and indexes, etc. Generally, the table data are stored in one or more shared storage spaces, each comprising one or more datafiles, although there are some RDBMS databases that store data for each table in a separate file, and others that store all of the data for a given database in a single file. Under Oracle's architecture, these shared storage spaces are called "tablespaces." Typically, an Oracle database will include a plurality tablespaces (e.g., system, user, rollback, etc.), wherein user data are stored in a plurality of tables in one or more selected tablespaces that may be specifically configured using various configuration parameters during creation or alteration of the tablespaces.

Data are stored in a tablespace in the following manner. First, a plurality of segments are allocated for the tablespace and a datafile is assigned to the tablespace, wherein each segment comprises a plurality of fixed-size (e.g., 2K, 4K or 8K) storage blocks. Each of these storage blocks comprise multiple operating storage blocks, which are the base unit the operating system uses to define where data is physically stored. These storage blocks are (typically) filled with data in a substantially sequential manner until close to all of the storage space provided by the allocated segments is consumed. At this point, the tablespace must be “extended” with one or more extents (similar to segments) so that additional data may be added. Typically, the data is logically stored in a plurality of rows, wherein each row of data includes a set of data (i.e., record) pertaining to various columns defined for the table the data are stored in. In Oracle, additional “row ID” information that uniquely identifies every row in the database is stored for each row. The row ID comprises a string having an encoded format that can be parsed by Oracle to quickly access the row of data corresponding to the row ID. The row IDs are used by primary key and other types of indexes to speed up queries and sorting operations.

As discussed above, the data are stored in a substantially sequential manner based on the approximate order the data are entered into a database. (For example, Oracle uses a background operation to write blocks of data in response to predetermined conditions (memory full, time interval, etc.), wherein the block writes are performed continuously, although they are slightly asynchronous to when data are actually entered.) As a result, the data for a given table are stored in a somewhat randomized order. For instance, suppose that contact information corresponding to various customers are stored in a table that includes various rows for storing the contact information, such as last name, first name, address, city,

state, phone number, etc. As new contact information is entered into the database, data pertaining to a new row will be written to the initial datafile (or currently active datafile) corresponding to the tablespace the table is assigned to. Since the contact information will usually not be entered in a sequential manner (e.g., alphabetically be last name), a sequential row-by-row examination of the data in the datafile will appear to not follow any predetermined ordering scheme, as illustrated by a randomized set of all contact records 18 in database 10.

In contrast to the foregoing data storage scheme, database users (both people operating client machines and internal software components that perform batch operations) typically desire to retrieve and/or view data in an ordered (i.e., sorted) manner. To meet this criteria, databases provide various solutions for providing requested data in a sorted configuration based on specified sort criteria, such as last name sorted alphabetically. In general, the data corresponding to a requested data set (i.e., query) must first be retrieved to a temporary sort space (comprising physical storage and/or memory space), whereupon the data are sorted corresponding to the predefined ordering scheme, and then provided to an application, applet, or module that is used to present the data to the user (or provide the data to an internal software component user).

In non-indexed queries (i.e., a query that doesn't use a primary key or other index for a table or set of related tables), a full table scan must be performed to retrieve appropriate rows specified by the query (the result set or recordset), and then a memory or disk sort must be performed on the result set prior to providing the data to the user as an "open" data set. For index-based queries, one or more indexes are used to identify the appropriate rows of data in the result set, whereupon mapping information provided by the index(es) (e.g., row IDs) that identifies where those rows of data are located is used to retrieve appropriate rows

of data meeting the query search criteria. In some instances, the rows of data are retrieved in a sorted order. In other cases, this is not possible or impractical, and the rows of data are first retrieved and then sorted in the sort space.

For example, suppose a user wanted to view all of the contacts stored in database 10 sorted by their last names. In the two-tier architecture illustrated in FIGURE 1, the user is running a client-side application 20 on client machine 14, which includes business logic 22 that generates a query 24 in response to a user request to view all of the contract records and submits query 24 to database 10 via network 16. Assuming the data are stored in a single CONTACT table that includes a LASTNAME column, the corresponding SQL query would look like:

```
SELECT * FROM CONTACTS
ORDER BY LASTNAME;
```

Generally, a request for all records in a table will cause a table scan to be performed, regardless of whether or not indexes are used. The rows in the contact table (stored as randomized set of contact records 18) are retrieved and sent to a sort area 26, which depending on the size of the sort may comprise a memory space and/or a temporary disk storage space. The rows are then sorted, using a sort process 28 and then provided back to client-side application 20 as a sorted full recordset 30.

The retrieval and sorting of data can be very time-consuming, particularly if the sorting has to be performed using a disk sort. This is often the case for large result sets, which may involve millions, or even 10's of millions of rows of data. In such instances, even an optimized query using indexes might take 10's of minutes or even hours. Furthermore, such queries are resource (CPU and memory) intensive, often slowing access by other users to a substantial halt.

In response to this problem, RDBMS database vendors have developed various schemes to provide data to GUI-based applications that function as client-

side front ends to enable users to access and view the database data. Typically, these schemes are centered around “virtual” lists of data that are either continually loaded using background operations or implemented through the use of built-in SQL commands, such as the TOP command. These conventional schemes are limited in their ability to retrieve and provide data to end users when large data sets are queried.

In continuance of the foregoing example, in one conventional embodiment, sorted full recordset 30 is logically stored as a virtual list that is managed by a virtual list manager 32. (In instances in which a very large number of rows are to be accessed, virtual list manager 32 may actually manage smaller sets of all of those rows that are continually being retrieved from database 10 using a background process – for convenience, a complete data set is illustrated in FIGURE 1.) Virtual list manager 32 interacts with a GUI manager 34 that is used to provide display information to a video subsystem that includes a display device (e.g., monitor) 36 to generate a GUI display screen 38 that enables the user to selectively view contact data stored in database 10. Typically, virtual list manager 32 will implement a filter 40 to pass a small number of rows of data to GUI manager 34, which then displays the rows in GUI display screen 38.

The conventional schemes for retrieving and display data from very large data sets have several problems. Most notably, they are very slow, consume a great deal of resources, and in the case of huge data sets, may not even be possible to perform in a useable manner.

## SUMMARY OF THE INVENTION

The present invention provides a method and system for accessing and displaying data records that are retrieved from large data sets in a manner that dramatically reduces the latency problems common to conventional access  
5 schemes. In accordance with the method, a large data set is logically partitioned into a plurality of "buckets" by defining "soft" boundaries corresponding to a sort order in which data records are presented to users or provided to internal users for purposes like batch processing. When a request to retrieve a data record or group of related records is received, e.g., in response to a user navigation event, the  
10 bucket of data the record or at least a portion of the group of related records is stored in is identified, and those records are retrieved from the database sorted in accordance with the sort order, and provided to the requestor.

Data defining a relative position of each boundary point in the sort order are built and stored as a set of boundary markers, preferably during an administrative  
15 process. Each of these boundary markers comprises a unique set of data corresponding to two or more columns in a database table on which the sort order is based, and in which at least a base portion of the data records are stored. Preferably, the chosen columns will already be indexed or have indexes applied to them subsequent to building the boundaries. In some instances, a pair of columns  
20 are used, such as columns that store first and last name data. When very large data sets are used, it may be necessary to use a tertiary set of column data, or even a higher number of columns. The boundary markers are stored in sets, preferably in accordance with the sort order they correspond to.

In response to a request to retrieve a data record or group of related data  
25 records, such as a list of contacts from a contact table or object having the same first and last name, the boundary markers are searched to determine which bucket



the desired data record or records are contained in. A query is then formulated based on the boundary markers to retrieve the records corresponding to the bucket.

In a typical n-tier implementation of the invention, the software components for implementing the invention are run on an application server that receives various data requests and navigation events from various clients. Upon retrieving an appropriate bucket of data records in response to such requests, all or a viewable subset (viewset) of the records are provided to the clients to enable users to access the data record or group of related data records. In instances in which viewsets are used, new viewsets of data records are provided to the client in response to navigation events, such as scrolling. In response to a navigation event that seeks data records that are not contained in a current bucket, a new bucket in which the requested data records are contained is identified and retrieved, and a new viewset from the new bucket is presented to the user. In this manners, users can search or browse data records retrieved from data sets that may comprise 10's of millions or even 100's of millions of records, wherein new viewsets are typically provided to the user with subsecond responses.

BRIEF DESCRIPTION OF THE DRAWINGS

The foregoing aspects and many of the attendant advantages of this invention will become more readily appreciated as the same becomes better understood by reference to the following detailed description, when taken in  
5 conjunction with the accompanying drawings, wherein:

FIGURE 1 is a schematic diagram illustrating how data records contained in a full data set are retrieved using conventional data access schemes found in the prior art;

FIGURE 2 is a schematic diagram conceptually illustrating how data records  
10 from large data sets are retrieved in "buckets" in accordance with the present invention;

FIGURE 3 is an exemplary architecture and data flow diagram corresponding to a two-tier implementation of the invention;

FIGURE 4 is an exemplary architecture and data flow diagram corresponding  
15 to a three-tier implementation of the invention;

FIGURE 5 is a flowchart illustrating the logic used by the invention when generating bucket boundaries;

FIGURE 6 is a flowchart illustrating the logic used by the invention in response to user navigation events; and

20 FIGURE 7 is a schematic diagram of a computer system that may be used for the client machines and servers in the architectures of FIGURES 3 and 4.

## DETAILED DESCRIPTION OF THE ILLUSTRATED EMBODIMENTS

A system and method for retrieval and displaying data from large datasets is described in detail herein. In the following description, numerous specific details are discussed to provide a thorough understanding of embodiments of the invention.

5 One skilled in the relevant art will recognize, however, that the invention can be practiced without one or more of the specific details, or with other methods, components, etc. In other instances, well-known structures or operations are not shown or described in detail to avoid obscuring aspects of various embodiments of the invention.

10 Reference throughout this specification to “one embodiment” or “an embodiment” means that a particular feature, structure, or characteristic described in connection with the embodiment is included in at least one embodiment of the present invention. Thus, the appearances of the phrases “in one embodiment” or “in an embodiment” in various places throughout this specification are not necessarily  
15 all referring to the same embodiment. Furthermore, the particular features, structures, or characteristics may be combined in any suitable manner in one or more embodiments.

The present invention provides a mechanism for accessing very large data sets, wherein the data sets are logically broken into manageable “buckets” of data,  
20 greatly reducing the data access latencies inherent in the prior art. The mechanism and its associated operations are referred to herein a large data volume (LDV) handling or simply LDV. LDV works by dividing a query that would normally be used to retrieve a large number of rows of data (e.g., contact data from a contact table or set of related tables) into a number of smaller queries, wherein each query specifies  
25 a set of data contained in a data *bucket*. A bucket is defined as all of the data returned from a corresponding SQL query, set with a pre-defined sort order, in which

upper and lower boundaries are used to specify the range of data contained in the bucket.

Logically, all of the rows of data for a target object (e.g., a table or set of tables containing related data) are sorted corresponding to a specified sort order and then divided into a plurality of contiguous buckets. This configuration is illustrated in FIGURE 2, in which a sorted contact table 50 is divided into a plurality of buckets 52 defined by boundaries 54. As used herein, the various data in a "base" table are logically shown as being divided alphabetically into buckets and corresponding bucket boundaries. This is merely illustrative to show the buckets and boundaries are generated corresponding to a sort order; as explained below, LDV provides various schemes for dividing data sets into buckets. Upon a request from an LDV user 56 to retrieve (for batch processing) or view (for end-users) a particular record or group of related records, LDV determines an appropriate bucket that record or group is stored in and retrieves the bucket. An appropriate query 58 is then formulated and submitted to database 10 for execution, returning a data set comprising a bucket 60 that includes data contained within a lower boundary 62 and upper boundary 64 specified by the query. The difference between the lower and upper boundaries define a size of the bucket 66. The bucket is then returned to LDV user 56, whereupon it may be used in a batch process or filtered by a client application that enables users to request and view data stored in database 10. In further response to various events, such as a user requesting to scroll past the end of the records contained in a current bucket or request to seek a record not contained in the current bucket, an appropriate query is formulated to retrieve a new bucket containing the desired records. In this manner, LDV operates like a sliding window 68 over the full set of data records contained in sorted contacts table 50.

An exemplary architecture 70 illustrating one implementation of the invention is shown in FIGURE 2. Architecture 70 includes several components that share the same root reference numerals (e.g., 10 and 10A, etc.) as components depicted in FIGURE 1 and discussed above; it will be understood that these components perform substantially similar operations in both architectures. Architecture 70 corresponds to a two-tier environment in which a client application 72 running on a client machine 14 enables a user to selectively view contact data stored in a datafile corresponding to a CONTACT table in database 10 as a randomized set of contact records 18A. It will be understood that in addition to the use of a single table for storing contact information, such information may be stored in a set of related tables, as will be recognized by those skilled in the database arts.

As discussed above, LDV works by retrieving buckets of data rather than an entire data set. The data contained in the buckets are logically bounded by boundaries, the generation of which are described below. In one embodiment, various boundary markers are stored in database 10A, such as depicted by a set of contact boundary markers 74. Optionally, all or a portion of the boundary markers may be stored in one or more files on database server 12A, or in one or more files on client machine 14A.

In one embodiment, various LDV operations are implemented by an LDV class 76 and an LDV helper class 78 provided by client application 72. As will be understood by those skilled in the art, client application 72 may comprise a single software module, or a set of modules that interact using appropriate application program interfaces (API's) 80. In response to user input 82, a business logic component 84 determines what data records a user of client machine 14 desires to view. Business logic component 84 then provides this information to LDV class 76, which generates a first query 86 to retrieve an appropriate set of boundary marker

data 88 corresponding to the object containing the data the user requested (in this instance the CONTACT table).

Upon receiving the boundary marker data, LDV class 76 parses the data to determine an appropriate pair of lower and upper boundaries corresponding to the bucket of data the requested data is contained in. For example, suppose that a user desires to search for the contacts with the last name of "Smith." Further suppose that a set of boundary data has been generated for the contact table that uses the LASTNAME column, and includes respective boundaries corresponding to the last record for selected letters C, H, K Q, U, and Z, as illustrated in FIGURE 3.

Accordingly, the boundary data are parsed to determine the nearest boundaries encompassing the row(s) of data containing a last name of "Smith," which in this case yield the boundaries at Q (lower boundary) and U (upper boundary) corresponding to a bucket 90. As a result, the following SQL query is formulated as a query 92 and submitted to database 10 to retrieve the bucket:

```
SELECT * FROM CONTACTS
WHERE LASTNAME >Q AND LASTNAME <=U
ORDER BY LASTNAME;
```

Generally, the columns used for generating boundaries will have a corresponding database index. As a result, in response to the query, database 10 uses a boundary index-based retrieval process 92 that retrieves all of the records for the CONTACTS table contained within the upper and lower boundaries for bucket 90, as depicted by all records having a LASTNAME beginning with the letters R, S, T, and U. The retrieved data are forwarded to sort area 26, whereupon they are sorted by sort process 28 and returned to client machine 14 as an open set comprising sorted bucket data 94. LDV class 76 (or another client-side module) then implements a display filter 96 that is used to determine a current subset of the sorted data in the bucket to be displayed (the "viewset) and passes the viewset to a

GUI manager 96, which drives the video subsystem to display data pertaining to the records in the viewset within a GUI display screen 38A on display device 36.

Another exemplary architecture 100 for implementing the invention in an n-tier environment is shown in FIGURE 4. As depicted, architecture 100 depicts a well-known 3-tier architecture in which an application server 102 “sits between” database server 12 and a plurality of client machines 14. As well be recognized by those skilled in the art, one or more additionally tiers (e.g., a web server tier) can be added to the illustrated 3-tier architecture.

Application server 102 hosts various software modules that are used to perform various operations, including delivering data to client machines 14 in response to corresponding requests for data from users of the machines. These components include an LDV class 76, and LDV helper class 78, which interact with a business logic component 104 via an API 106. In one embodiment, client machines 14 host a client application 108 that includes a business logic component 110 and a GUI manager 112.

In general, the various aspects of the LDV technology are performed in a substantially similar manner in both of architectures 70 and 100. However, the interaction between the various other software components differ, as follows. A typical request process begins with user input 114 that is received by business logic 110. The business logic processes the user input to determine what data the user desires and submits a corresponding data request 116 to application server 102. In one embodiment, business logic components 110 and 104 comprises a client-server set, that are configured to interact with one another using a common protocol. Accordingly, data requests comprising “native” data pertaining to the common protocol are sent from business logic component 110 to business logic

component 104. In another embodiment, client application 108 comprises a web browser, and data request 116 comprises HTML (hyper-text markup language) data.

Typically, data request 116 will be received by business logic 104. Optionally, data request 116 will be received by LDV class 76. The LDV class and LDV helper class then interact with database 10 in a similar manner to that described above, returning a sorted data bucket 94 to the application server. The LDV class and/or business logic 104 then determine what data to filter, ultimately sending the data from business logic component 104 to client machine 14 as filtered display data 118. As before, the filtered display data may be in a native format, or comprise HTML data. Filtered display data is then passed by business logic 110 to GUI manager 112, which drives the client machine's video subsystem to generate a GUI display screen 36B on display device 36.

#### Building Buckets

As discussed above, LDV retrieves selected buckets of data based on data requests from users rather than entire data sets. The data contained in a given bucket is defined by the boundaries for the bucket. Generally, the boundaries are pre-determined through an administrative process that is somewhat similar to building indexes on a database. Alternately, the boundaries may be generated at runtime.

LDV boundaries are generated in the following manner. Initially, parameters for the process must be selected. With reference to FIGURE 5, this process begins in a block 150 in which a table is selected to apply LDV boundaries to. Typically, these tables will store data that is usually presented to end users in a sorted list format, such as contacts, products, employees, clients, etc. In many instances, the tables will comprise "base" or "parent" tables, while the displayed data will include not only data contained in the base or parent table, but also data stored in one or



more child tables that are linked to the base or parent table via a primary key-foreign key relationship. For the purposes of LDV, only the base or parent table needs to be considered when generating LDV boundaries.

Next, in a block 152, allowable sort orders for the table are determined.

- 5 Generally, the sort orders will be based on the intended use of the data, and in one embodiment will be enforced through the LDV data access classes and subsequently by the graphical user interface. Each sort order to be provided requires its own unique set of boundary markers. For example, suppose the targeted table is a contact table that includes contact data stored in various columns, including
- 10 LASTNAME, FIRSTNAME, COMPANY, ADDRESS, CITY, STATE, etc. Typically, it will be desired to sort the contact data alphabetically by LASTNAME. In addition, other sort orders may be included, such as alphabetical sorted on COMPANY, CITY, STATE etc. Furthermore, the sort order may be complex, comprising a combination of columns. For example, in one instance the data might be sorted on
- 15 LASTNAME, then FIRSTNAME, then COMPANY. In another instance, the data may be sorted by COMPANY, then LASTNAME, then FIRSTNAME. The actual combination used will depend on the type of data, the column configurations that data are stored in, and how the data are to be presented to the user (or to be used internally by, e.g., a batch processing component). Finally, the selected column or
- 20 columns should be columns that are either currently indexed, or will have a database index generated for them prior to using the LDV scheme.

- The following operations are then performed for each sort order, as indicated by start and end loop blocks 154 and 156. First, in a block 158 the approximate number of records to be included within each bucket is determined. To determine
- 25 this value, many considerations are required. First, how will the data generally be requested and used? In a two-tier environment for heavy batch processing, a larger

bucket size is acceptable, since the entire data set will often need to be processed and the speed of a two-tier connection is provided. In an n-tier environment for GUI restricted usage, like that used in customer relationship management (CRM) data systems, a small bucket size would be better since only a small amount of the data (e.g., 20 – 30 rows) is viewed at once. Furthermore, since there are many users who are concurrently access the database in typical data systems of this type, larger bucket sizes should be avoided because of their higher server overhead.

Another consideration is “viewability” of data. As a general rule, the size of the bucket should be scaled inversely with the percentage of the database that the user sees. For example, if users always use selection criteria that restrict their view to half of the records in the database, then the bucket size should be doubled. This is because, if a user has a selection criteria set that eliminates half of the database from their view, then each bucket will contain only one-half of its capacity of records that match the baseline selection criteria (when compared with a baseline condition in which the boundaries are defined as if all records will be access). As a result, the actual number of records returned in the bucket remain substantially constant

Initialized through an administrative operation, the parameters are used by a process that opens the data set sorted in the appropriate order, as provided by a block 160. In practice, data corresponding to only those columns considered in the sort order need be retrieved, rather than the entire records. For example, if the sort order is based on LASTNAME, then FIRSTNAME, the corresponding SQL query might look like:

```
SELECT LASTNAME, FIRSTNAME FROM CONTACTS
ORDER BY LASTNAME, FIRSTNAME;
```

The process then loops through the open data set in a block 162, start to finish, defining boundary markers based on data found every *N* rows, wherein *N* corresponds to the number of records for the bucket determined in block 158.

Depending on the data size and uniqueness of the data in the chose column, a single column may not be unique enough to mark a boundary. In these cases, the scheme enhances the boundary definition through a second and/or tertiary column, incorporating that data into the boundary marker. For example, suppose the

5 LASTNAME, FIRSTNAME, and COMPANY columns are considered for use as boundary markers. In a small data set of approximately 5,000 rows, each contact's last name may be unique (or have limited duplications), so only data corresponding to the LASTNAME column need to be used for the boundary markers. With large

10 data volumes, it may be common for various data values in the LASTNAME column to be repeated more than N times, such that that two or more sequential boundary markers would be identical if only the LASTNAME column was considered. In this instance, the boundary markers should include data from a second column, such as LAST NAME. In extremely large databases, these two columns may also not be sufficient, so data in a tertiary column (e.g., COMPANY) will be necessary.

15 Similarly, this can be taken to a fourth or higher order if necessary to ensure appropriate uniqueness for each boundary marker. Optionally, a unique data column could be used, such as a client ID column or the like.

When complete, the boundary marker data are stored as a set so that it can be easily accessed during production data retrieval situations. For example, these

20 data may be stored in a database table on database 10, on application server 102, or locally on the client machines. In one embodiment, the boundary marker data is stored as a concatenated string within an optimized database table. In accordance with a tertiary marker comprising LASTNAME, FIRSTNAME, and COMPANY data, boundary markers are stored in a concatenated string having the following format:

25 ...|Smith~~Frank~~Boeing|Smith~~Gerald~~Tyco|Smith~~Harold~~IBM|... (1)

In this format, each adjacent set of boundary marker data (e.g., "[Smith ~~ Frank ~~ Boeing]") corresponds to an adjacent boundary point.

In another embodiment, each set of boundary marker data may be stored as a separate record in a table. Whether an implementation uses the concatenated string format or separate records (or still other configurations that will be recognized by those skilled in the data system arts) will likely be dependent on the relative workload of the database and application servers. If the database server workload is light to moderate, built-in database operations (e.g., SQL and cursor loops) can be used to search for appropriate boundary markers. The advantage of the concatenated string technique is that the entire string can be passed to the application server (which requires very minimal overhead for the database server), and then parsed at the application server rather than having an equivalent operation performed by the database server.

As discussed above, building bucket boundaries for a given data set requires the data set to be opened. Conveniently, opening the data set can be performed using the LDV scheme itself so that the entire data set does not need to be opened (at one time) to complete the administrative process. As a result, building bucket boundaries can be carried on during 7x24 operation of the database, without adversely affecting enterprise applications that access the database. In the event that the very first run of the boundary building process is performed, appropriate seed data (i.e., estimated boundaries that should satisfactorily break up the data) can easily be implemented to define a temporary set of boundaries that may be used to retrieve records from the data set using temporary buckets, whereby the actual boundaries and corresponding markers are determined by looping through each temporary bucket of records.

#### Using Boundaries to Generate SQL for Data Buckets

Each bucket of data that is retrieved from the database includes all of the records contained within the lower and upper boundaries corresponding to the data bucket. Also, any given record in the entire data set will be contained in only one bucket. Accordingly, when using the boundary marker string discussed above, determination of the boundary markers corresponding to the bucket in which a particular data record is stored is performed by parsing the string until passing the first boundary marker that would be logically ordered prior to the particular data record. For example, if the search was for a "Gilbert Smith," concatenated string (1) would be parsed until the "|Smith~~Gerald~~Tyco|" portion of the string was encountered. This represents the lower boundary marker. The next set of boundary marker data corresponds to upper boundary marker. These are used to formulate the SQL statement for the bucket.

In a simple boundary definition that only uses a single column, the SQL will include data in the where clause corresponding to that column. For example, suppose a user is searching on the last name of "Billingsby" in a CONTACT table and the upper and lower boundaries respectively comprise "Benton" and "Bingham." The corresponding SQL statement would then be:

```
SELECT * FROM CONTACTS
WHERE LASTNAME >'Benton' AND LASTNAME <= 'Bingham'
ORDER BY LASTNAME;
```

A simple boundary definition, as described above, works well for thousands of rows, but can breakdown when there are many entries with the same value in an extremely large database. An example is a list of names that contains 200,000 Smith's. A more complex algorithm is required for that case. In this instance, the boundary markers will be based on multiple database columns to further break the data into appropriate buckets. In the name example, by using a secondary boundary based on the FIRSTNAME column and a tertiary boundary on the

COMPANY column, hundreds of millions of records can be handled since there is a very limited chance of data grouping (i.e., relatively large groups of records having identical values for LASTNAME, FIRSTNAME, and COMPANY). Using the bucket boundary data, the resulting query can be simple to very complex depending on boundaries. A simple bucket would be created when there are many Fred Smiths in the list:

```
SELECT * FROM CONTACTS
WHERE LASTNAME = 'Smith' AND FIRSTNAME = 'Fred' AND
COMPANY >= 'Avis' AND COMPANY < 'Tyco'
ORDER BY COMPANY;
```

A more complex query results when a bucket spans the first name:

```
SELECT * FROM CONTACTS
WHERE LASTNAME = 'Smith' AND FIRSTNAME = 'Fred' AND
COMPANY >= 'Xerox' OR
LASTNAME = 'SMITH' AND FIRSTNAME > 'Fred' AND
FIRSTNAME < 'Funkie'
ORDER BY FIRSTNAME, COMPANY;
```

An even more complex query can result when the bucket spans a last name as well.

Although the SQL can appear to be quite complex, resulting query plans show very quick isolation of data due to the user of indexed columns according to the specified sort order defined by the corresponding boundary marker set.

#### Scrolling Through Data Buckets

When implemented, the LDV scheme enables users to navigate through various data screens or "views" during which appropriate buckets are retrieved and data contained in those buckets are displayed in a manner that is transparent to the users. A flowchart illustrating the logic used by the invention to implement this functionality is shown in FIGURE 6.

Initially, a query will be performed to retrieve a first bucket of data based on a requested search, as provided by a block 170 in FIGURE 6. When a typical data

set is opened, the data returned includes a sorted set of all of the data matching the SQL query criteria, with record positioned at the beginning of the data set being marked as the active record. With LDV applied, the returned record set includes all of the data falling within the first bucket, positioned on the first record, and has

5 knowledge that the record set is an "incomplete" set. As the bucket data is navigated (e.g., browsed or scrolled through using various user inputs), appropriate viewsets (i.e., subsets of the bucket that are displayed at any one time to the user) are presented to the user. As used herein, each user navigation input (e.g., page-up, page down, etc.) will generate a new navigation "context" for the user, and an

10 appropriate new viewset will be provided to the user. These operations are depicted by blocks 172 and 174 in the flowchart. In effect, the viewset acts as a sliding window over the sorted records in the bucket. At the same time, the LDV classes monitor regular record set calls corresponding to navigation events such as MoveNext, MovePrevious and EOF, to determine if the user desires to view data

15 that is not contained in the current bucket. If this condition is sensed, as indicated by a YES (TRUE) result to a decision block 176, appropriate boundaries for the new bucket are determined in a block 178. In one embodiment, the current bucket is discarded in a block 180. Optionally, the data corresponding to the current bucket may be maintained in a cache, such as a LRU (least recently used) cache. The

20 choice of whether to discard buckets or use a cache will depend generally depend on the latency characteristics of the bucket retrievals.

Next, in a block 182, the new bucket of data is retrieved from the database, whereupon an appropriate viewset is sent to the client and the active record is positioned to meet the user's request. For example, if the user request data for

25 "Frank Smith" the active record will correspond to the first "Frank Smith" encountered in the sorted list.

Sparsely Populated or Restricted Record Sets

Generally, the entire set is open when the GUI allows for an unfiltered list of data or when a batch process is going to operate on all of the data within a table. However, filters are often applied to the sorted columns of the query in order to return a targeted set of data. Analysis of the additional SQL and monitoring of the retrieved data sets allows the LDV helper class to alter the way it selects bucket boundaries to more optimally return data. For example, if a 'user' only requires people with a last name starting with "S", the helper processing immediately recognizes the additional restriction on the sort order columns and begins its bucket boundaries at the first boundary pair that includes that data. For example, the additional SQL (indicated in bold) for the first bucket might be:

```
SELECT * FROM CONTACTS
WHERE LASTNAME LIKE 'S*'
AND LASTNAME => 'S' AND
LASTNAME < 'Sampson'
ORDER BY LASTNAME;
```

If the query is filtered on a column not covered by the LDV scheme or is part of an external table join, the pre-query processing to determine a bucket boundary cannot be made. However, through monitoring of the returned data sets, the LDV helper class can alter the number of buckets returned by one query. In the simplest form, this is accomplished by skipping a boundary definition and moving to the next one. For example, suppose a user selects everyone in New York City from a database that contains data on every person in the United States. This would result, assuming evenly distributed last names throughout the US, in about 1/25<sup>th</sup> of the expected rows per bucket, since approximately one in every twenty-five Americans live in New York City. In an attempt to retrieve the preset optimal count per bucket, the helper class would monitor the number of records returned in response to an initial or prior query, and internally increase the number of skipped



boundaries in subsequent queries to retrieve new buckets of data as scrolling continues until the general returned amount is close to a desired bucket size. In this example, the helper might observe that the number of records retrieved via a first query resulted in only  $1/25^{\text{th}}$  of the number of records in the desired bucket size, whereupon the helper may adjust subsequent queries to skip 25 boundaries per re-query. The end result of this extra processing reduces unnecessary trips to the database server during a scrolling operation. This added technology is known as Dynamic Bucketing.

#### Rapid Seek Mechanism

When displaying a list of records, there are two main functional needs that must be met: The ability to soft seek on a piece of data, and the ability to re-open the application properly positioned within the list. Although trivial with small amounts of data, these become very complex tasks when dealing with hundreds of millions of rows. LDV bucketing information allows these tasks to be accomplished in sub-second time frames.

Using the boundary data, a soft seek forward in the sorted list becomes a simple task. The LDV class finds the appropriate bucket that the search for piece of data will lie in, returns that bucket of data (typically a sub-second call), and positions the active record on the appropriate record using its regular seek process. All the overhead of opening, sorting and seeking through hundreds of millions of rows is avoided.

To restart an application on a certain record, a unique key is usually known – such as the client ID. To take advantage of the boundary data, the application stores the client ID (or other unique identifier) for the last records that was active and simply needs to query a single row corresponding to the stored client ID to return to the record (very quick). The appropriate sort column data can then be

extracted from the record, enabling an appropriate bucket containing the record to be identified, whereupon the record set corresponding to the bucket is retrieved and opened (typically a sub-second call), and the active record is positioned on the record corresponding to the stored client ID.

5       The present invention provides several advantages over conventional large data set access techniques. Most notably, data access latencies are greatly reduced, often enabling a user to literally scroll through millions of records without requiring the user to wait for data to be retrieved or screens to be updated. The LDV scheme easily scales from small data sets to very large data sets, and enables  
10 a large number of concurrent users to leverage its features, while causing minimal additional overhead for both the database server and application servers.

#### Exemplary Computer System for Practicing the Invention

With reference to FIGURE 7, a generally conventional computer 200 is illustrated, which is suitable for use as client machines, application servers, and  
15 database servers in connection with practicing the present invention, and may be used for running client and server-side software comprising one or more software modules that implement the various operations of the invention discussed above. Examples of computers that may be suitable for client machines as discussed above include PC-class systems operating the Windows NT or Windows 2000  
20 operating systems, Sun workstations operating the UNIX-based Solaris operating system, and various computer architectures that implement LINUX operating systems. Computer 200 is also intended to encompass various server architectures, as well as computers having multiple processors.

Computer 200 includes a processor chassis 202 in which are mounted a  
25 floppy disk drive 204, a hard drive 206, a motherboard 208 populated with appropriate integrated circuits including memory 210 and one or more processors

(CPUs) 212, and a power supply (not shown), as are generally well known to those of ordinary skill in the art. It will be understood that hard drive 206 may comprise a single unit, or multiple hard drives, and may optionally reside outside of computer 200. A monitor 214 is included for displaying graphics and text generated by software programs and program modules that are run by the computer. A mouse 216 (or other pointing device) may be connected to a serial port (or to a bus port or USB port) on the rear of processor chassis 202, and signals from mouse 216 are conveyed to the motherboard to control a cursor on the display and to select text, menu options, and graphic components displayed on monitor 214 by software programs and modules executing on the computer. In addition, a keyboard 218 is coupled to the motherboard for user entry of text and commands that affect the running of software programs executing on the computer. Computer 200 also includes a network interface card 220 or built-in network adapter for connecting the computer to a computer network, such as a local area network, wide area network, or the Internet.

Computer 200 may also optionally include a compact disk-read only memory (CD-ROM) drive 222 into which a CD-ROM disk may be inserted so that executable files and data on the disk can be read for transfer into the memory and/or into storage on hard drive 206 of computer 200. Other mass memory storage devices such as an optical recorded medium or DVD drive may be included. The machine instructions comprising the software that causes the CPU to implement the functions of the present invention that have been discussed above will likely be distributed on floppy disks or CD-ROMs (or other memory media) and stored in the hard drive until loaded into random access memory (RAM) for execution by the CPU. Optionally, all or a portion of the machine instructions may be loaded via a computer network.

Although the present invention has been described in connection with a preferred form of practicing it and modifications thereto, those of ordinary skill in the art will understand that many other modifications can be made to the invention within the scope of the claims that follow. Accordingly, it is not intended that the scope of the invention in any way be limited by the above description, but instead be  
5 determined entirely by reference to the claims that follow.

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000  
1001  
1002  
1003  
1004  
1005  
1006  
1007  
1008  
1009  
1010  
1011  
1012  
1013  
1014  
1015  
1016  
1017  
1018  
1019  
1020  
1021  
1022  
1023  
1024  
1025  
1026  
1027  
1028  
1029  
1030  
1031  
1032  
1033  
1034  
1035  
1036  
1037  
1038  
1039  
1040  
1041  
1042  
1043  
1044  
1045  
1046  
1047  
1048  
1049  
1050  
1051  
1052  
1053  
1054  
1055  
1056  
1057  
1058  
1059  
1060  
1061  
1062  
1063  
1064  
1065  
1066  
1067  
1068  
1069  
1070  
1071  
1072  
1073  
1074  
1075  
1076  
1077  
1078  
1079  
1080  
1081  
1082  
1083  
1084  
1085  
1086  
1087  
1088  
1089  
1090  
1091  
1092  
1093  
1094  
1095  
1096  
1097  
1098  
1099  
1100  
1101  
1102  
1103  
1104  
1105  
1106  
1107  
1108  
1109  
1110  
1111  
1112  
1113  
1114  
1115  
1116  
1117  
1118  
1119  
1120  
1121  
1122  
1123  
1124  
1125  
1126  
1127  
1128  
1129  
1130  
1131  
1132  
1133  
1134  
1135  
1136  
1137  
1138  
1139  
1140  
1141  
1142  
1143  
1144  
1145  
1146  
1147  
1148  
1149  
1150  
1151  
1152  
1153  
1154  
1155  
1156  
1157  
1158  
1159  
1160  
1161  
1162  
1163  
1164  
1165  
1166  
1167  
1168  
1169  
1170  
1171  
1172  
1173  
1174  
1175  
1176  
1177  
1178  
1179  
1180  
1181  
1182  
1183  
1184  
1185  
1186  
1187  
1188  
1189  
1190  
1191  
1192  
1193  
1194  
1195  
1196  
1197  
1198  
1199  
1200  
1201  
1202  
1203  
1204  
1205  
1206  
1207  
1208  
1209  
1210  
1211  
1212  
1213  
1214  
1215  
1216  
1217  
1218  
1219  
1220  
1221  
1222  
1223  
1224  
1225  
1226  
1227  
1228  
1229  
1230  
1231  
1232  
1233  
1234  
1235  
1236  
1237  
1238  
1239  
1240  
1241  
1242  
1243  
1244  
1245  
1246  
1247  
1248  
1249  
1250  
1251  
1252  
1253  
1254  
1255  
1256  
1257  
1258  
1259  
1260  
1261  
1262  
1263  
1264  
1265  
1266  
1267  
1268  
1269  
1270  
1271  
1272  
1273  
1274  
1275  
1276  
1277  
1278  
1279  
1280  
1281  
1282  
1283  
1284  
1285  
1286  
1287  
1288  
1289  
1290  
1291  
1292  
1293  
1294  
1295  
1296  
1297  
1298  
1299  
1300  
1301  
1302  
1303  
1304  
1305  
1306  
1307  
1308  
1309  
1310  
1311  
1312  
1313  
1314  
1315  
1316  
1317  
1318  
1319  
1320  
1321  
1322  
1323  
1324  
1325  
1326  
1327  
1328  
1329  
1330  
1331  
1332  
1333  
1334  
1335  
1336  
1337  
1338  
1339  
1340  
1341  
1342  
1343  
1344  
1345  
1346  
1347  
1348  
1349  
1350  
1351  
1352  
1353  
1354  
1355  
1356  
1357  
1358  
1359  
1360  
1361  
1362  
1363  
1364  
1365  
1366  
1367  
1368  
1369  
1370  
1371  
1372  
1373  
1374  
1375  
1376  
1377  
1378  
1379  
1380  
1381  
1382  
1383  
1384  
1385  
1386  
1387  
1388  
1389  
1390  
1391  
1392  
1393  
1394  
1395  
1396  
1397  
1398  
1399  
1400  
1401  
1402  
1403  
1404  
1405  
1406  
1407  
1408  
1409  
1410  
1411  
1412  
1413  
1414  
1415  
1416  
1417  
1418  
1419  
1420  
1421  
1422  
1423  
1424  
1425  
1426  
1427  
1428  
1429  
1430  
1431  
1432  
1433  
1434  
1435  
1436  
1437  
1438  
1439  
1440  
1441  
1442  
1443  
1444  
1445  
1446  
1447  
1448  
1449  
1450  
1451  
1452  
1453  
1454  
1455  
1456  
1457  
1458  
1459  
1460  
1461  
1462  
1463  
1464  
1465  
1466  
1467  
1468  
1469  
1470  
1471  
1472  
1473  
1474  
1475  
1476  
1477  
1478  
1479  
1480  
1481  
1482  
1483  
1484  
1485  
1486  
1487  
1488  
1489  
1490  
1491  
1492  
1493  
1494  
1495  
1496  
1497  
1498  
1499  
1500  
1501  
1502  
1503  
1504  
1505  
1506  
1507  
1508  
1509  
1510  
1511  
1512  
1513  
1514  
1515  
1516  
1517  
1518  
1519  
1520  
1521  
1522  
1523  
1524  
1525  
1526  
1527  
1528  
1529  
1530  
1531  
1532  
1533  
1534  
1535  
1536  
1537  
1538  
1539  
1540  
1541  
1542  
1543  
1544  
1545  
1546  
1547  
1548  
1549  
1550  
1551  
1552  
1553  
1554  
1555  
1556  
1557  
1558  
1559  
1560  
1561  
1562  
1563  
1564  
1565  
1566  
1567  
1568  
1569  
1570  
1571  
1572  
1573  
1574  
1575  
1576  
1577  
1578  
1579  
1580  
1581  
1582  
1583  
1584  
1585  
1586  
1587  
1588  
1589  
1590  
1591  
1592  
1593  
1594  
1595  
1596  
1597  
1598  
1599  
1600  
1601  
1602  
1603  
1604  
1605  
1606  
1607  
1608  
1609  
1610  
1611  
1612  
1613  
1614  
1615  
1616  
1617  
1618  
1619  
1620  
1621  
1622  
1623  
1624  
1625  
1626  
1627  
1628  
1629  
1630  
1631  
1632  
1633  
1634  
1635  
1636  
1637  
1638  
1639  
1640  
1641  
1642  
1643  
1644  
1645  
1646  
1647  
1648  
1649  
1650  
1651  
1652  
1653  
1654  
1655  
1656  
1657  
1658  
1659  
1660  
1661  
1662  
1663  
1664  
1665  
1666  
1667  
1668  
1669  
1670  
1671  
1672  
1673  
1674  
1675  
1676  
1677  
1678  
1679  
1680  
1681  
1682  
1683  
1684  
1685  
1686  
1687  
1688  
1689  
1690  
1691  
1692  
1693  
1694  
1695  
1696  
1697  
1698  
1699  
1700  
1701  
1702  
1703  
1704  
1705  
1706  
1707  
1708  
1709  
1710  
1711  
1712  
1713  
1714  
1715  
1716  
1717  
1718  
1719  
1720  
1721  
1722  
1723  
1724  
1725  
1726  
1727  
1728  
1729  
1730  
1731  
1732  
1733  
1734  
1735  
1736  
1737  
1738  
1739  
1740  
1741  
1742  
1743  
1744  
1745  
1746  
1747  
1748  
1749  
1750  
1751  
1752  
1753  
1754  
1755  
1756  
1757  
1758  
1759  
1760  
1761  
1762  
1763  
1764  
1765  
1766  
1767  
1768  
1769  
1770  
1771  
1772  
1773  
1774  
1775  
1776  
1777  
1778  
1779  
1780  
1781  
1782  
1783  
1784  
1785  
1786  
1787  
1788  
1789  
1790  
1791  
1792  
1793  
1794  
1795  
1796  
1797  
1798  
1799  
1800  
1801  
1802  
1803  
1804  
1805  
1806  
1807  
1808  
1809  
1810  
1811  
1812  
1813  
1814  
1815  
1816  
1817  
1818  
1819  
1820  
1821  
1822  
1823  
1824  
1825  
1826  
1827  
1828  
1829  
1830  
1831  
1832  
1833  
1834  
1835  
1836  
1837  
1838  
1839  
1840  
1841  
1842  
1843  
1844  
1845  
1846  
1847  
1848  
1849  
1850  
1851  
1852  
1853  
1854  
1855  
1856  
1857  
1858  
1859  
1860  
1861  
1862  
1863  
1864  
1865  
1866  
1867  
1868  
1869  
1870  
1871  
1872  
1873  
1874  
1875  
1876  
1877  
1878  
1879  
1880  
1881  
1882  
1883  
1884  
1885  
1886  
1887  
1888  
1889  
1890  
1891  
1892  
1893  
1894  
1895  
1896  
1897  
1898  
1899  
1900  
1901  
1902  
1903  
1904  
1905  
1906  
1907  
1908  
1909  
1910  
1911  
1912  
1913  
1914  
1915  
1916  
1917  
1918  
1919  
1920  
1921  
1922  
1923  
1924  
1925  
1926  
1927  
1928  
1929  
1930  
1931  
1932  
1933  
1934  
1935  
1936  
1937  
1938  
1939  
1940  
1941  
1942  
1943  
1944  
1945  
1946  
1947  
1948  
1949  
1950  
1951  
1952  
1953  
1954  
1955  
1956  
1957  
1958  
1959  
1960  
1961  
1962  
1963  
1964  
1965  
1966  
1967  
1968  
1969  
1970  
1971  
1972  
1973  
1974  
1975  
1976  
1977  
1978  
1979  
1980  
1981  
1982  
1983  
1984  
1985  
1986  
1987  
1988  
1989  
1990  
1991  
1992  
1993  
1994  
1995  
1996  
1997  
1998  
1999  
2000  
2001  
2002  
2003  
2004  
2005  
2006  
2007  
2008  
2009  
2010  
2011  
2012  
2013  
2014  
2015  
2016  
2017  
2018  
2019  
2020  
2021  
2022  
2023  
2024  
2025  
2026  
2027  
2028  
2029  
2030  
2031  
2032  
2033  
2034  
2035  
2036  
2037  
2038  
2039  
2040  
2041  
2042  
2043  
2044  
2045  
2046  
2047  
2048  
2049  
2050  
2051  
2052  
2053  
2054  
2055  
2056  
2057  
2058  
2059  
2060  
2061  
2062  
2063  
2064  
2065  
2066  
2067  
2068  
2069  
2070  
2071  
2072  
2073  
2074  
2075  
2076  
2077  
2078  
2079  
2080  
2081  
2082  
2083  
2084  
2085  
2086  
2087  
2088  
2089  
2090  
2091  
2092  
2093  
2094  
2095  
2096  
2097  
2098  
2099  
2100  
2101  
2102  
2103  
2104  
2105  
2106  
2107  
2108  
2109  
2110  
2111  
2112  
2113  
2114  
2115  
2116  
2117  
2118  
2119  
2120  
2121  
2122  
2123  
2124  
2125  
2126  
2127  
2128  
2129  
2130  
2131  
2132  
2133  
2134  
2135  
2136  
2137  
2138  
2139  
2140  
2141  
2142  
2143  
2144  
2145  
2146  
2147  
2148  
2149  
2150  
2151  
2152  
2153  
2154  
2155  
2156  
2157  
2158  
2159  
2160  
2161  
2162  
2163  
2164  
2165  
2166  
2167  
2168  
2169  
2170  
2171  
2172  
2173  
2174  
2175  
2176  
2177  
2178  
2179  
2180  
2181  
2182  
2183  
2184  
2185  
2186  
2187  
2188  
2189  
2190  
2191  
2192  
2193  
2194  
2195  
2196  
2197  
2198  
2199  
2200  
2201  
2202  
2203  
2204  
2205  
2206  
2207  
2208  
2209  
2210  
2211  
2212  
2213  
2214  
2215  
2216  
22